AA

Computer Security

2nd Edition
Expanded & Updated

Practical UNIX & Internet Security

O'REILLY®

Simson Garfinkel and Gene Spafford

Practical UNIX & Internet Security

Garfinkel & Spafford

Practical UNIX and Internet Security, Second Edition
by Simson Garfinkel and Gene Spafford

Published by O'Reilly & Associates, Inc., 101 Morris Street, Sebastopol, CA 95472.

Editor: Deborah Russell

Production Editor: Nicole Gipson Arigo

Printing History:

June 1991:	First Edition. Copyrighted under the title *Practical UNIX Security*.
September 1991:	Minor corrections.
April 1992:	Minor corrections.
October 1992:	Minor corrections.
March 1993:	Minor corrections.
August 1993:	Minor corrections.
June 1994:	Minor corrections.
April 1996:	Second Edition. Completely rewritten and expanded to include Internet security.

Nutshell Handbook, the Nutshell Handbook logo, and the O'Reilly logo are registered trademarks and The Java Series is a trademark of O'Reilly & Associates, Inc.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and O'Reilly and Associates, Inc. was aware of a trademark claim, the designations have been printed in caps or initial caps.

While every precaution has been taken in the preparation of this book, the authors and publisher assume no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.

ISBN: 1-56592-148-8
[M] [11/00]

Table of Contents

10

Auditing and Logging

After you have established the protection mechanisms on your system, you will need to monitor them. You want to be sure that your protection mechanisms actually work. You will also want to observe any indications of misbehavior or other problems. This process of monitoring the behavior of the system is known as *auditing*. It is part of a defense-in-depth strategy: to borrow a phrase from several years ago, you should trust, but you should also verify.

UNIX maintains a number of log files that keep track of what's been happening to the computer. Early versions of UNIX used the log files to record who logged in, who logged out, and what they did. Newer versions of UNIX provide expanded logging facilities that record such information as files that are transferred over the network, attempts by users to become the superuser, electronic mail, and much more.

Log files are an important building block of a secure system: they form a recorded history, or *audit trail*, of your computer's past, making it easier for you to track down intermittent problems or attacks. Using log files, you may be able to piece together enough information to discover the cause of a bug, the source of a break-in, and the scope of the damage involved. In cases where you can't stop damage from occurring, at least you will have some record of it. Those logs may be exactly what you need to rebuild your system, conduct an investigation, give testimony, recover insurance money, or get accurate field service performed.

But beware: Log files also have a fundamental vulnerability. Because they are often recorded on the system itself, they are subject to alteration or deletion. As we shall see, there are techniques that may help you to mitigate this problem, but no technique can completely remove it unless you log to a different machine.

Locating to a different machine is actually a good idea even if your system supports some other techniques to store the logs. Consider some method of automatically sending log files to a system on your network in a physically secured location. For example, sending logging information to a PC or Apple Macintosh provides a way of storing the logs in a machine that is considerably more difficult to break into and disturb. We have heard good reports from people who are able to use "outmoded" 80486 or 80386 PCs as log machines. For a diagram of such a setup, see Figure 10-1.

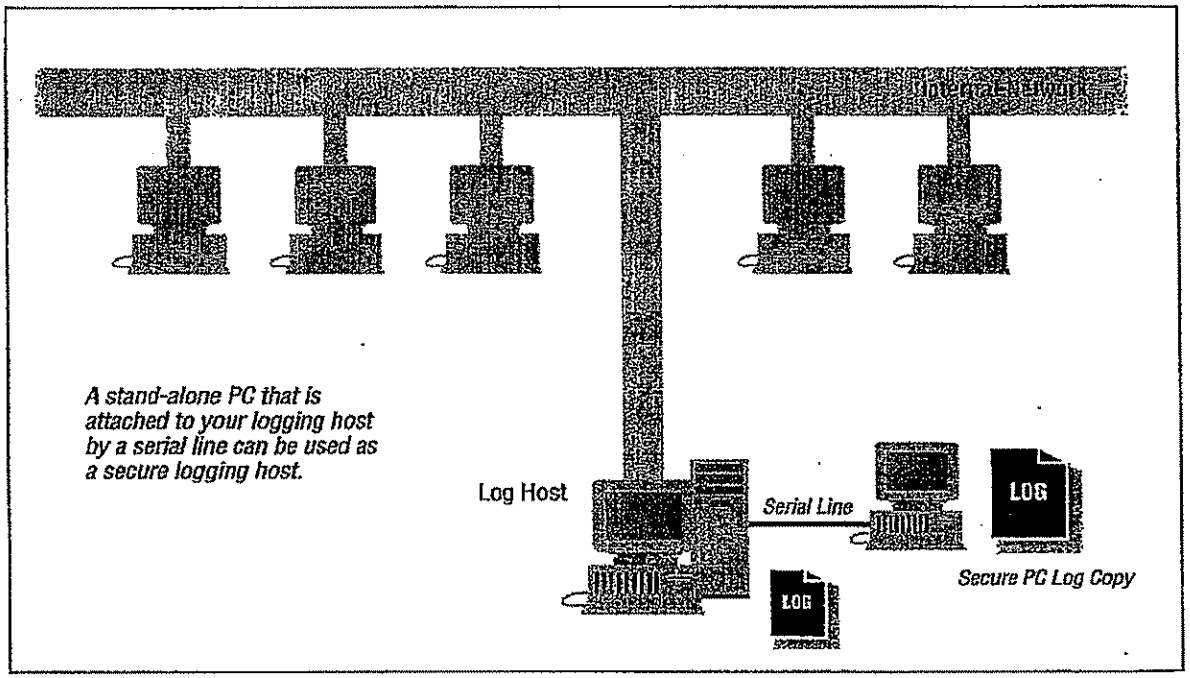


Figure 10-1. Secure logging host.

The Basic Log Files

Most log files are text files that are written line by line by system programs. For example, each time a user on your system tries to become the superuser by using the *su* command, the *su* program may append a single line to the log file *sulog*, which records whether the *su* attempt was successful or not.

Different versions of UNIX store log files in different directories. The most common locations are:

/usr/adm	Used by early versions of UNIX
/var/adm	Used by newer versions of UNIX, so that the */usr* partition can be mounted read-only
/var/log	Used by some versions of Solaris, Linux, BSD, and free BSD to store log files

Within one of these directories (or a subdirectory in one of them) you may find variants of some or all of the following files:

acct or *pacct*	Records commands run by every user
aculog	Records of dial-out modems (automatic call units)
lastlog	Logs each user's most recent successful login time, and possibly the last unsuccessful login too
loginlog	Records bad login attempts
messages	Records output to the system's "console" and other messages generated from the *syslog* facility
sulog	Logs use of the *su* command
utmp[1]	Records each user currently logged in
utmpx	Extended *utmp*
wtmp[2]	Provides a permanent record of each time a user logged in and logged out. Also records system shutdowns and startups
wtmpx	Extended *wtmp*
vold.log	Logs errors encountered with the use of external media, such as floppy disks or CD-ROMs
xferlog	Logs FTP access

[1] *Most versions of UNIX store the utmp file in the /etc directory.*
[2] *Early versions of System V UNIX stored the wtmp file in the /etc directory*

The following sections describe some of these files and how to use the UNIX *syslog* facility.

lastlog File

UNIX records the last time that each user logged into the system in the *lastlog* log file. This time is displayed each time you log in:

```
login: ti
password: books2sell
Last login: Tue Jul 12 07:49:59 on tty01
```

C2 Audit

Many UNIX systems allow the administrator to enable a comprehensive type of auditing (logging) known as "C2 audit." This is so-named because it is logging of the form specified by U.S. Department of Defense regulations to meet the certification at the C2 level of trust. These regulations are specified in a document called the *Trusted Computer System Evaluation Criteria* (often referred to as the "Orange Book" in the "Rainbow Series").

C2 auditing generally means assigning an *audit ID* to each group of related processes, starting at login. Thereafter, certain forms of system calls performed by every process are logged with the audit ID. This includes calls to open and close files, change directory, alter user process parameters, and so on.

Despite the mandate for the general content of such logging, there is no generally accepted standard for the format. Thus, each vendor that provides C2-style logging seems to have a different format, different controls, and different locations for the logs. If you feel the need to set such logging on your machine, we recommend that you read the documentation carefully. Furthermore, we recommend that you be careful about what you log so as not to generate lots of extraneous information, and that you log to a disk partition with lots of space.

The last suggestion, above, reflects one of the biggest problems with C2 audit: it can consume a huge amount of space on an active system in a short amount of time. The other main problem with C2 audit is that it is useless without some interpretation and reduction tools, and these are not generally available from vendors—the DoD regulations only require that the logging be done, not that it be usable! Vendors have generally provided only as much as is required to meet the regulations and no more.

Only a few third-party companies provide intrusion detection or audit-analysis tools: *Stalker*, from Haystack Laboratories, is one such product with some sophisticated features. Development of more sophisticated tools is an ongoing, current area of research for many people. We hope to be able to report something more positive in a third edition of this book.

In the meantime, if you are not using one of these products, and you aren't at a DoD site that requires C2 logging, you may not want to enable C2 logging (unless you like filling up your disks with data you may not be able to interpret). On the other hand, if you have a problem, the more logging you have, the more likely you will be able to determine what happened. Therefore, review the documentation for the audit tools provided with your system if it claims C2 audit capabilities, and experiment with them to determine if you want to enable the data collection.

What's a Firewall?

A *firewall* gives organizations a way to create a middle ground between networks that are completely isolated from external networks, such as the Internet, and those that are completely connected. Placed between an organization's internal network and the external network, the firewall provides a simple way to control the amount and kinds of traffic that will pass between the two.

The term *firewall* comes from the construction industry. When apartment houses or office buildings are built, they are often equipped with firewalls—specially constructed walls that are resistant to fire. If a fire should start in the building, it may burn out of control in one portion, but the firewall will stop or slow the progress of the fire until help arrives.

The same philosophy can be applied to the protection of local area networks of machines from outside attack. Used within an organization, a firewall can limit the amount of damage: an intruder may break into one set of machines, but the firewall will protect others. Erected between an organizational network and the Internet at large, a firewall prevents a malicious attacker who has gained control of computers outside the organization's walls from gaining a foothold on the inside. Firewalls seem to make sense because there is always a "fire" burning somewhere on the Internet.

Default Permit vs. Default Deny

The fundamental function of a firewall is to restrict the flow of information between two networks. To set up your firewall, you must therefore define what kinds of data pass and what kinds are blocked. This is called defining your firewall's *policy*. After a policy is defined, you must then create the actual *mechanisms* that implement that policy.

There are two basic strategies for defining firewall policy:

Default permit
: With this strategy, you give the firewall the set of conditions that will result in data being blocked. Any host or protocol that is not covered by your policy will be passed by default.

Default deny
: With this strategy, you describe the specific protocols that should be allowed to cross through the firewall, and the specific hosts that may pass data and be contacted. The rest are denied.

There are advantages and disadvantages to both default permit and default deny. The primary advantage of default permit is that it is easier to configure: you

simply block out the protocols that are "too dangerous," and rely on your awareness to block new dangerous protocols as they are developed (or discovered). With default deny, you simply enable protocols as they are requested by your users or management. Any protocol that isn't being used by your organization might as well be blocked.

Neither default permit nor default deny is a panacea. With both policies, you can create a firewall that is either secure or unsecure, by permitting (or failing to deny) "dangerous" protocols.

Uses of Firewalls

Firewalls are part of a good defense in depth strategy. The idea is to place several layers of protection between your machines and the potential threats. There are some obvious threats from the outside, so you should naturally place a firewall between the outside and your internal network(s).

Because a firewall is placed at the intersection of two networks, it can be used for many other purposes besides simply controlling access. For example:

- Firewalls can be used to block access to particular sites on the Internet, or to prevent certain users or machines from accessing certain servers or services.
- A firewall can be used to monitor communications between your internal network and an external network. For example, you could use the firewall to log the endpoints and amount of data sent over every TCP/IP connection between your organization and the outside world.
- A firewall can even be used to eavesdrop and record all communications between your internal network and the outside world. A 56KB leased line at 100% utilization passes only 605 MB/day, meaning that a week's worth of Internet traffic can easily fit on a single 8mm digital tape. Such records can be invaluable for tracking down network penetrations or detecting internal subversion.*
- If your organization has more than one physical location and you have a firewall for each location, you can program the firewalls to automatically encrypt packets that are sent over the network between them. In this way, you can use the Internet as your own private wide area network (WAN) without compromising the data; this process is often referred to as creating a *virtual private network*, or VPN. (You will still be vulnerable to traffic analysis and denial of service attacks, however.)

* Such records also pose profound privacy questions, and possibly legal ones as well. Investigate these questions carefully before engaging in such monitoring.

EE

THIS EXHIBIT HAS BEEN REDACTED IN ITS ENTIRETY

FF

MAI 3

SOFT COMPUTING

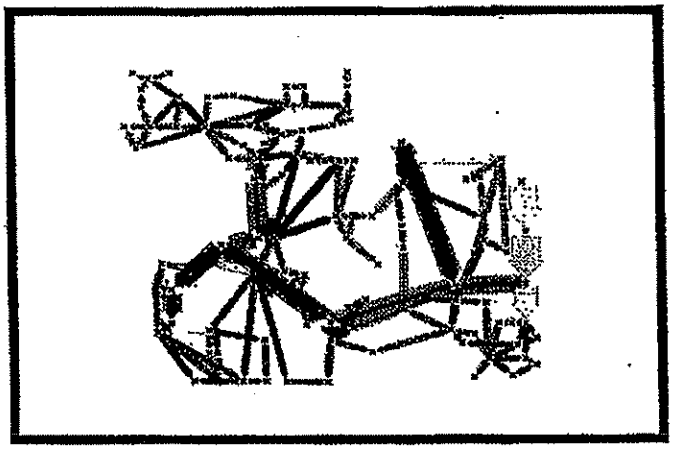
Rough Sets

Fuzzy Logic

Neural Networks

Uncertainty Management

Knowledge Discovery



Edited by
T. Y. Lin
and
A. M. Wildberger

SCS SPONSORED BY
THE SOCIETY FOR COMPUTER SIMULATION
ISBN 1-56555-077-3

Statistical Methods for Computer Usage Anomaly Detection Using NIDES (Next-Generation Intrusion Detection Expert System)

Alfonso Valdes and Debra Anderson (SRI International) January 27, 1995

Abstract : With the dramatic growth of computer networks in recent months, the need to protect the integrity of information assets from unauthorized use has never been greater. SRI's Next Generation Intrusion Detection Expert System (NIDES), a fielded system for computer system monitoring, has both an expert system component and a statistical component integrated into a client-server model and communicating to a system security officer via a window-oriented user interface. The statistical component of NIDES, NIDES/STAT, is a nonparametric anomaly detection subsystem that has been successfully used to identify suspicious behavior on the part of computer users as well as UNIX applications. NIDES/STAT learns profiles of usage behavior and then flags deviations of short-term behavior from its more slowly varying historical profiles without knowing *a priori* scenarios of computer misuse and without constructing any sort of discriminant function between subjects. Results from recent studies indicate that NIDES/STAT is quite successful in detecting abnormal patterns in application usage data.

1 Introduction

With the growing concern over security of computer systems, several organizations have developed methods to automatically detect computer usage that is possibly improper or unauthorized. One such system, SRI's NIDES (Next Generation Intrusion Detection Expert System) [1], examines audit trail information using a custom nonparametric statistical component as well as a rulebased component, and is capable of processing audit records in real time or batch mode. The statistical component, NIDES/STAT, generates an anomaly score for each audit record by comparing recent observations with a long-term or historical profile which NIDES learns for each subject. Recent observations are maintained in a short-term profile, which is an integrated summary of a subject's activity typically spanning one to a few hundred audit records (intended to reflect minutes on a typical UNIX system).

The NIDES paradigm models a computer system as a set of subjects who initiate actions that affect objects. In this paradigm explored by SRI [1, 2, 5, 6], subjects can include computer users, applications, processes, network hosts, and so forth. To date, we have successfully employed NIDES to detect anomalous use with both computer users and application programs as subjects. Herein, we present an overview of the NIDES/STAT methodology and the results of an experiment profiling UNIX applications with NIDES.

2 NIDES Statistical Algorithm Description

The statistical approach used in NIDES [3] is to compare a subject's short-term behavior with its historical or long-term behavior, considering both long-term behavior absent from short-term behavior, and short-term behavior atypical of long-term behavior. Whenever short-term behavior is sufficiently unlike long-term behavior, a warning flag is raised. In general, short-term behavior is somewhat different from long-term behavior, because short-term behavior is more concentrated on specific activities and long-term behavior is distributed across many activities. To accommodate this expected deviation, the NIDES statistical component keeps track of the amount of deviation that it has seen in the past and issues a warning only if this deviation exceeds a subject-specific threshold.

The actual processing is as follows. The statistical component receives audit data either in real time or from files. This audit data serves both to score current behavior and to train profiles, with the former operation only possible after a period of initial profile training. As each audit record is received, NIDES modifies a fading memory summary of the recent activity with the activity observed on the newly arrived record. The fading memory concept is implemented by exponential aging of various summary counts. NIDES also maintains summaries of all the activity since the last long-term profile update. The long-term profile is updated once per day in real-time operation, or when the audit data stream timestamps cross a date boundary in batch processing. Updating consists of exponential fading of the existing long-term profile and combining the summary of activity maintained since the last update.

The NIDES statistical approach requires no *a priori* knowledge about what type of behavior would result in compromised security. It simply compares short-term and long-term behaviors to determine whether they are statistically similar. This feature of the NIDES statistical component makes it ideally suited for the task of computer usage anomaly detection in the absence of intrusion scenarios.

2.1 Profile Training

NIDES describes subject behavior by means of a profile, which we separate into short-term and long-term components. Anomaly scoring compares counts in a short-term profile with expected counts (which are based on historical probabilities maintained in a long-term profile) by means of a normalized square difference measure that is computationally chi-square like in form. Aspects of subject behavior are represented as measures (characterizations of usage along such dimensions as file access, CPU usage, hour of use, and so forth). The observed difference is compared on a measure per measure basis with the empirical distribution of the historically observed difference, from which we obtain a half-normal deviate that is now comparable across all measures. The squares of these are summed and compared to historically determined thresholds.

NIDES profile training consists of three phases: NIDES first learns the historical probabilities with which various categories are observed, then NIDES learns the empirical distribution of the deviation of short-term observations about these long-term category distributions, and finally NIDES sets the score thresholds. The short-term profile changes every audit record, while the long-term profile changes at regularly scheduled profile updates. Exponential fading of both profiles (using different time constants) permits relatively compact representation of the profiles as well as a mechanism for forgetting activity from the distant past not repeated in the recent past. Exponential fading of the short-term profile takes place as each audit record is received, while fading of the long-term profile takes place at profile update time. As a consequence of this aging mechanism, we speak of effective counts when referring to a count (for example, the number of times a category has been observed) to which this aging has been applied.

NIDES uses four classes of measures: activity intensity, audit record distribution, categorical, and continuous. The activity intensity measures determine whether the volume of activity generated is normal. The audit record distribution measure determines whether, for recently observed activity, the types of actions being generated are normal. The categorical and continuous measures examine whether, within a type of activity (say, accessing a file), the types of observed categories are typical. Categorical measures are those for which the observed values are by nature categorical. For example, the file access measure would have as its categories the names of accessed files. Continuous measures are those for which the observed values are numeric, such as CPU usage. For each measure, we construct a probability distribution of short-term and long-term behaviors. For example, for the measure of file access, the long-term probability distribution would consist of the historical probabilities with which different files have been accessed, and the short-term probability distribution would consist of the recent probabilities with which different files have been accessed. In this case, the categories to which probabilities are attached are the file names. In the case of continuous measures, such as CPU time, the categories to which probabilities are attached are ranges of values, which we refer to as bins (these ranges of values are mutually exclusive and span all values from 0 to infinity). The bins are scaled multiplicatively so that the range of values is logarithmically assigned with the ratio of the first to the last bin endpoint being approximately 1000. For example, if 10 bins are available, the right endpoint of the 10th is chosen to be the data mean plus three to four standard deviations, and then each bin endpoint proceeding downward is obtained by halving the next higher. This gives a factor of ten powers of two (or 1024), as desired. The binning procedure uses fractional powers of two for any number of bins other than ten. This mechanism is sufficiently robust that the scaling parameter need not be precisely estimated. By the application of this procedure, NIDES transforms continuous measures to categorical from the standpoint of its internal computations.

Category counts and probabilities are maintained as follows. For each measure, we define the following parameters:

- H_{Neff}: historical effective n
- P_i: historical probability, category i
- $Count_i$: arithmetic count since the last long-term profile update of the number of observations of category i
- $Agecount_i$: count of observations of category i since last update, with fading
- $NCATS$: number of categories for the measure

In addition, we define the following global parameters:

- γ_s: short-term fading factor
- γ: long-term fading factor

We then describe the processing for each measure. Suppose that on the current audit record, category i is observed for the measure of interest. The aged and unaged counts in the short-term profile are adjusted by aging the existing counts of all categories for the measure (using the short-term aging factor) and incrementing the

count of the observed category. Algorithmically, this consists of the following steps:

$$Count_i = Count_i + 1 \ ,$$

$$Agecount_j = \gamma_s \times Agecount_j, j \neq i \ ,$$

$$Agecount_i = \gamma_s \times Agecount_i + 1 \ .$$

The *Count* field is used to modify the long-term profile at the next update interval as follows. At each update time, the counts of observations since the last long-term profile update across all categories for a given measure are totaled into "today's count" (*TodayCount* below). The historical effective n (H_{Neff}) is aged by multiplying with the long-term aging factor γ_l. The contents of the long-term profile are converted from probabilities to effective counts by multiplying each historical bin probability P_i by H_{Neff}. The counts are then combined with the counts accumulated since the last update time $Count_i$, and converted back to probabilities (dividing by the new aged count). The algorithm for the updating is outlined below:

$$TodayCount = \sum_{i=1}^{NCATS} Count_i \ ,$$

$$H_{Neff} = \gamma_l * H_{Neff} \ ,$$

$$TempCount_i = H_{Neff} \times P_i + Count_i \ ,$$

After doing the above for all categories i, the historical count is updated and the totals converted to probabilities:

$$H_{Neff} = H_{Neff} + TodayCount \ ,$$

$$P_i = TempCount_i / H_{Neff} \ .$$

After these calculations, the category probabilities are examined to see if they should be dropped or grouped into a rare class. NIDES has mechanisms for dropping categories that fall below some threshold probability and grouping as rare those categories above the drop threshold but with sufficiently small probabilities that they might affect the statistical stability of the algorithms.

2.2 Differences Between Long- and Short-term Profiles — The Q Statistic

The *Agecount* field is used for estimating the difference statistic for each audit record. The degree of difference between the long-term profile for a measure and the short-term profile for a measure is quantified using a chi-square-like statistic, with the long-term profile playing the role of the actual probability distribution and the short-term profile playing the role of the observations. We call the resultant numerical value Q; there is a different Q value for each measure, updated as each audit record is encountered. Large values of Q mean that the long-term and short-term profiles for the measure are very different from one another and therefore warrant some suspicion; a Q value of zero means that they agree exactly.

We let N_{eff} denote the short-term effective n, mathematically defined as

$$N_{eff} = \sum_{i=0}^{Nobs} \gamma_s^i \ .$$

This quantity has an asymptotic value of $\frac{1}{1-\gamma_s}$, although the algorithm uses the actual value given by the above expression. The calculation of the Q statistic proceeds as follows:

$$e_i = N_{eff} \times P_i \ ,$$

$$Q = \sum_{i=1}^{NCats} \frac{(e_i - AgeCount_i)^2}{e_i} \ .$$

The reader may note that Q is similar to a chi-square random variable. Unfortunately, it is not possible to refer Q directly to a chi-square table due to potential dependence and insufficient observations for some bins in the data stream on which Q is based. Since the distribution of Q is not chi-squared, we need to track its values to determine what its distribution looks like. We observe the values for Q (computed on each audit record) and build an empirical probability distribution for Q using an aging and updating mechanism similar to that used for the measure categories. There is a Q statistic and a corresponding Q distribution for each measure. The Q distributions look somewhat like long-tailed and stretched-out chi-square distributions. Let QP_j be the empirical probability that Q falls in bin j of its distribution, and let TP_j be the corresponding tail probability, obtained as

$$TP_j = \sum_{k=j}^{NBins} QP_k \ .$$

2.3 Scoring Anomalous Behavior — The S and $T2$ Statistic

We transform the tail probability for Q and denote the transformed variable as S, defining the transformation so that S has a half-normal distribution. (A half-normal distribution looks like the right-hand side of a normal

distribution, except that the height of the probability distribution is doubled so that there is still unit area under the curve. This is also the distribution of the absolute value of a normally distributed variable.) The mapping from tail probabilities of the Q distribution to half-normal values is obtained by interpolation from a table of the tail of a normal distribution. Mathematically, the mapping from a tail probability to an S value takes the form

$$S = \Phi^{-1}(1 - TP/2).$$

As each audit record is received, we observe the bin in the Q distribution into which the computed value of Q falls, extract the corresponding tail probability value, and generate the corresponding S value according to the above equation. This is repeated for all measures, resulting in a vector of S values. High S values correspond to measures that are unusual relative to the typical amount of discrepancy that occurs between long-term and short-term profiles. Small S values correspond to measures that are not unusual relative to the amount of discrepancy that typically occurs between long-term and short-term profiles. We combine the S scores into an overall statistic that we call $T2$. This statistic is a summary judgment of the abnormality of all active measures, and is given by the sum of the squares of the S statistics normalized by the number of measures:

$$T2 = \frac{\sum_{m=1}^{Nmeas} S_m^2}{Nmeas}$$

As is the case with Q, we build a long-term distribution for $T2$ rather than rely on a parametric model to obtain threshold values. The long-term distribution for $T2$ is built during the last stage of the profile building period, after reasonably stable long-term distributions for the Q statistics have been constructed. We declare recent audit records to be anomalous at the yellow or warning level whenever the $T2$ value is over the 1% threshold value of the long-term empirical distribution for $T2$, and at the red or critical level whenever the $T2$ is over the 0.1% threshold.

3 Detection of Anomalous Behavior in Application Usage

SRI adapted NIDES/STAT to detect masquerading applications from exit records extracted from UNIX audit data [4]. We examined approximately three months of application use in a UNIX environment, spiked with records from two applications representing abnormal usage, with four exit records for one application and one for another. We attempted to detect these records against the trained profiles from 26 legitimate applications. We observed 101 detections in 130 opportunities, corresponding to a detection rate of approximately 77%. This is the detection rate for a single execution of a masquerading program; the probability of eventually detecting masquerader activity from several executions of the program is much higher. For example, with this detection rate, the probability of detecting at least one of two executions of a masquerading program is approximately 95%. In addition to detection performance, any system such as NIDES must also achieve an acceptably low false positive rate, defined as the percentage of detections for a subject processed through its own profile. The observed false positive rate for legitimate applications for this experiment was 1.3%, based on observations not used in the training set (the nominal false positive rate was configured to be 1%). Table 1 gives the false positive results for our experiment, as well as a summary of the detection results for the masquerader applications.

3.1 Cross-profiling Experiment

Cross profiling is the term we use when running one subject's audit data through another subject's long-term profile. In such an experiment, we use the terms host to denote the application whose profile is being used and guest to denote the application supplying the data. Cross profiling allows us to determine how unique an application's profile is and how successful other applications might be in trying to masquerade as the host application. By examining the detection rate from cross profiling we can also assess the similarity of profiles among subjects with related functionality, with an eye to constructing group profiles. Grouping may be used to provide default initial profiles for subjects based on their group membership, allowing for a faster "bootstrapping" of the NIDES profile training mechanism.

Comparing the detection rates between applications shows three possible relationships: asymmetric detection, where an application can pass through the profile of another application (a low detection rate), but the other application cannot easily pass through the profile of the original application; mutually low detection, where pairs or small groups of applications can mutually pass through each other's profiles; and mutual detection, where for a pair of applications neither can pass through the other's profile without raising suspicion.

Table 2 shows the minimum, maximum, and average detection rates using the yellow threshold (1%) for each application. Subjects with high average detection rates, such as `getfullnm` and `latex`, are very sensitive

Application Profiling Results			
	False Positive		
	Yellow	Red	Detections
as	0.0	0.0	+++*
cat	3.9	1.9	_** *
compile	0.0	0.0	-+* *
cp	0.0	0.0	—* *
csh	0.5	0.5	+++* *
discuss	0.7	0.0	**++ *
emacs	2.0	0.3	**++ +
finger	0.0	0.0	-+* *
fmt	0.3	0.0	+*** *
gawk	1.3	0.0	++++ *
getfullnm	2.6	1.3	**** *
ghostview	0.9	0.0	**++ *
grep	0.1	0.0	—+ *
latex	3.9	0.0	**** *
less	0.7	0.4	+++* *
ls	1.0	0.1	—* *
mail	0.0	0.0	++++ *
make	2.2	0.0	+-+ *
man	0.9	0.0	-+* *
more	0.7	0.0	-++* *
mymoreproc	0.8	0.0	+*** *
pwd	0.4	0.4	**** *
rm	0.3	0.0	-+* *
sort	1.1	0.0	-++ *
stty	0.0	0.0	++** *
vi	1.3	0.1	—— *
Total *			62
Total +			39
Total -			29

This table summarizes the false-positive and detection results for the application profiling study. The false positive column shows two percents: the percent of observations above the yellow (nominally 1%) detection threshold, and the percent above the red (nominally 0.1%) threshold. The column labeled "Detections" gives the result of processing each masquerading record through the host application's profile. We have recorded an asterisk (*) for detection above the critical (red) threshold, a plus (+) for detection above the warning (yellow) threshold, and a dash (-) for no detection. The groupings indicate the results for the four instances of the first masquerader followed by the single instance of the second. At the bottom of the table are total counts of the number of red (*), yellow (+), and non-detections (-).

Table 1: Application Profiling Results

to potential masquerader data. Others, with low average rates, such as `vi` and `grep`, are more tolerant and would be candidates for a masquerading attempt. The detection thresholds for `getfullnm` and `latex` are somewhat low, while those for `vi` and `grep` are on the high side. This may explain the low detection rate for masquerader data using `vi` as a host profile and confirms our low detection rates for `vi` under our true-positive tests in the first three experiments.

	Detection Percentages(%)		
Application	Minimum	Maximum	Average
as	0.10	100	53.62
cat	0.00	98.84	52.13
compile	0.90	98.33	24.71
cp	0.00	98.77	24.07
csh	0.00	99.17	42.76
discuss	7.04	99.49	72.87
emacs	10.53	98.80	86.23
finger	3.37	99.91	59.23
fmt	34.82	100.00	90.02
gawk	14.76	100.00	75.15
getfullnm	71.54	99.98	98.24
ghostview	5.92	99.31	82.21
grep	0.00	90.94	13.38
latex	94.62	99.81	98.53
less	1.02	99.69	48.04
ls	0.00	87.87	32.84
mail	1.15	99.68	64.27
make	1.70	96.83	51.27
man	5.88	99.96	63.32
more	0.16	86.85	35.06
mymoreproc	3.29	100.00	82.40
pwd	29.64	99.74	91.82
rm	0.00	97.01	34.81
sort	6.84	99.32	82.96
stty	49.59	99.76	95.43
vi	0.00	36.97	6.19

This table shows the minimum, maximum, and average detection percents for all applications processed through the profile of the host (row) application. For example, across all subjects, the average detection rate of the as profile was 53.62%.

Table 2: Detection Results for Cross-Profiling Experiment

4 Conclusions

We have presented a summary of the NIDES statistical methodology (NIDES/STAT) and the result of using this methodology to profile UNIX applications. NIDES/STAT is embedded in SRI's NIDES, an inte-

grated system for computer anomaly detection incorporating NIDES/STAT, a rule-based component, and a graphic user interface. NIDES/STAT does not rely on parametric models or some inter-subject distance function. It learns subject behavior by observing this behavior over time, and scores new behavior according to its similarity to past behavior. The methodology does not depend on any models of inappropriate computer use. Based on recent experimental results using actual UNIX audit data, NIDES/STAT is a powerful detector, correctly classifying 77% of records representing inappropriate usage while experiencing a false positive rate of 1.3%. It is evident that NIDES/STAT can successfully detect such records as well as distinguish between legitimate subjects. The proven performance of NIDES establishes it as a leading tool for those wishing to ensure the integrity of computer systems.

Acknowledgments:

Our research was supported by the U.S. Navy who funded SRI under contract N00039-92-C-0015 and by Trusted Information Systems through contract F30602-91-C-0067 which was funded by the U.S. Air Force, Rome Laboratory.

References

[1] D. Anderson, T. Frivold, A. Tamaru, A Valdes. NIDES User Manual/Computer System Operators Manual — Beta Release. Technical Report, Computer Science Laboratory, SRI International, Menlo Park, California, June 1994.

[2] R. Jagannathan, T. F. Lunt, F. Gilham, A. Tamaru, C. Jalali, P. Neumann, D. Anderson, T. D. Garvey, and J. Lowrance. Requirements Specification: Next Generation Intrusion Detection expert system(NIDES). Technical Report, Computer Science Laboratory, SRI International, Menlo Park, California, September 1992.

[3] H. S. Javitz and A. Valdes. The NIDES Statistical Component: Description and Justification. Technical Report, Computer Science Laboratory, SRI International, Menlo Park, California, March 1994.

[4] D. Anderson, T. Lunt, H. S. Javitz, A. Tamaru, A. Valdes. Detecting Unusual Program Behavior Using the NIDES Statistical Component. Technical Rreport, Computer Science Laboratory, SRI International, Menlo Park, California, December 1993.

[5] D. Anderson, T. Frivold, A. Tamaru, A. Valdes. Next Generation Intrusion Detection Expert System (NIDES) Software Design Specifications. Technical Report, Computer Science Laboratory, SRI International, Menlo Park, California, July 1994.

[6] T. F. Lunt, Ann Tamaru, Fred Gilham, R. Jagannathan, Caveh Jalali, H. S. Javitz, A. Valdes, P. G. Neumann, and T. D. Garvey. A Real-time Intrusion Detection Expert System (IDES), Final Technical Report, Computer Science Laboratory, SRI International, Menlo Park, California, February 1992.

KK

THIS EXHIBIT HAS BEEN REDACTED IN ITS ENTIRETY